IM 1003: Programming Design

File I/O and C++ Strings

Ling-Chieh Kung

Department of Information Management
National Taiwan University

April 28, 2014

Applications of classes

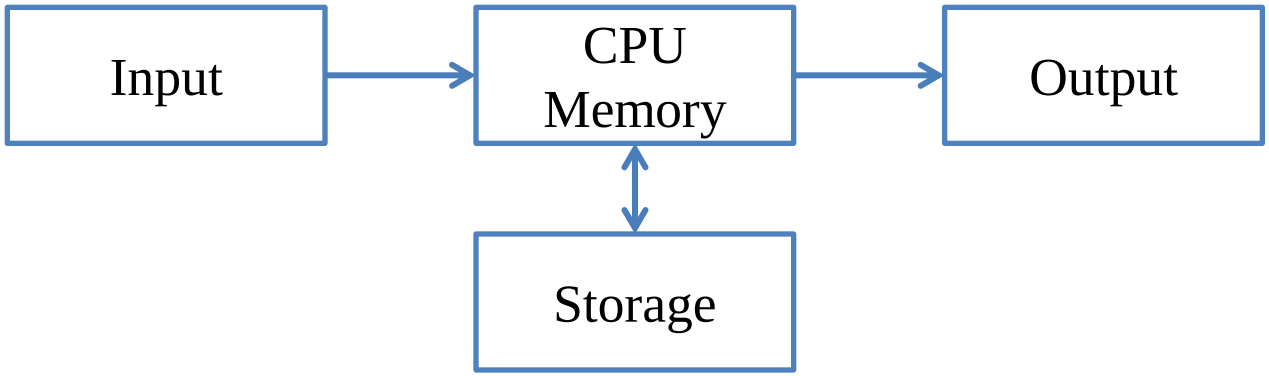
- We have studied a lot about classes.
  - Encapsulation.
  - Constructors, copy constructors, destructors.
  - Operator overloading.
- Remaining topics:
  - Inheritance.
  - Polymorphism.
- Today let's study two applications of classes.
  - File input/output.
  - C++ strings.

Outline

- **File I/O**
  - **Writing data to a file**
  - Reading data from a file
- C++ Strings

File I/O

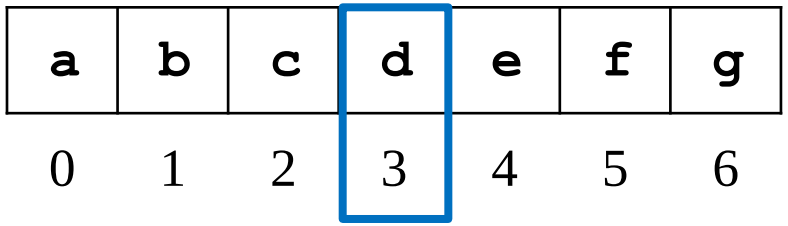
- The **von Neumann architecture**:



- With the techniques of **file input/output** (file I/O), we will read data from and store data to files in the **hard discs**.
  - So that the results can still be kept **after** the program **terminates**.
- We will focus on **plain-text files**.
  - Those files that can be directly edited with Notepad on MS Windows.

A plain-text file

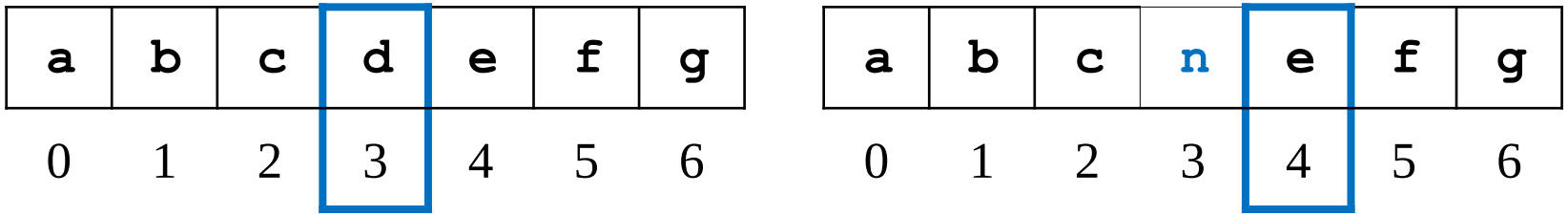
- Files store data.
  - A plain-text file stores **characters**.
  - A MS Word document stores characters and **format** information.
  - A bitmap file stores **color** codes.
- How are characters stored in a plain-text files?
  - Each character has its own **position**.
  - For each opened file, there is a **position pointer** indicating the **current reading/writing position**.



  - To control the reading/writing operations, we control the position pointer.

Writing to a file

- The first character is stored at **position 0**.
- In general, once a character is written to a file:
  - The character replaces the old character at the **current** position.
  - The position pointer moves to the **next** position (from *i* to *i* + 1).
- When a character **`n`** is written to this file:



File streams

- In C++, input and output activities are managed in **streams**.
  - E.g., data may flow from **`cin`** or into **`cout`**.
- To replace the console and keyboard by files, in C++ we create **`ifstream`** and **`ofstream`** objects.
- **`ifstream`** and **`ofstream`** are classes defined in **`<fstream>`**.
  - They can be used to create input/output file stream objects.
  - Simply imagine those objects as target files!

Output file streams

- To open and close an **output file stream**:

```
ofstream file object;
file object.open(file name);
// ...
file object.close();
```

```
ofstream myFile;
myFile.open("temp.txt");
// ...
myFile.close();
```

  - **open()** and **close()** are **public member functions**.
  - ***file name*** is a C string.
- Is there a member variables storing the file name?
- How are **open()** and **close()** implemented?

Writing to an output file stream

- To write to an output file stream, we may use **<<**.

```
ofstream myFile;
myFile.open("temp.txt");
myFile << "1 abc\n &%^ " << 123.45;
myFile.close();
```

  - **<<** has been **overloaded** for the class **ofstream**.
  - It returns **ofstream&** for concatenated output streams.
  - The second argument of **<<** can be of any basic data type.
- What if we want to put a **MyVector** object as the second argument?
- What if we replace **myFile** by **cout** in the third statement.

Options for an output file stream

- An **open mode** can be set when we open an output file stream.

```
ofstream file object;
file object.open(file name, option);
// ...
file object.close();
```

  - **ios::out** (default): The window starts at location 0; remove existing data.
  - **ios::app**: The window starts at the end; never modify existing data.
  - **ios::ate**: The window starts at the end; can modify existing data.
- **ios** is a class; **out**, **app**, and **ate** are **public static variables**.

Constructors and other members

- The class **ofstream** also provide **constructors**:

```
ofstream file object (file name, option);
```

```
ofstream file object (file name);
```

```
ofstream myFile("temp.txt");
myFile << "1 abc\n &%^ " << 123.45;
myFile.close();
```

  - Regardless of the extension name, we are creating/opening a plain text file.
- **ofstream** provides other member functions.
  - E.g., **put(char c)** writes the character **c** into the file.

Example

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

int main()
{
  ofstream scoreFile("temp.txt", ios::out);
  char name[20] = {0};
  int score = 0;
  char notFin = 0;
  bool con = true;
  if(!scoreFile)
    exit(1);
  while (con)
  {
    cin >> name >> score;
    scoreFile << name << " " << score << "\n";
    cout << "Continue (Y/N)? ";
    cin >> notFin;
    con = ((notFin == 'Y') ? true : false);
  }
  scoreFile.close();
  return 0;
}
```

- What will happen if we replace **`scoreFile`** by **`cout`**?
- How to check whether a

Outline

- **File I/O**
  - Writing data to a file
  - **Reading data from a file**
- C++ Strings

Input file streams

- To read data from a file, we create an input file stream.
- We create an **ifstream** object.

```
ifstream file object;
file object.open(file name);
// ...
file object.close();
```

```
ifstream myFile;
myFile.open("temp.txt");
// ...
myFile.close();
```

- The only open mode we will use for **ifstream** is **iso::in** (default).
- Again, we may use **if(!myFile)** to check whether a file is really opened.
  - If the file does not exist, **!myFile** returns false.

Reading from an input file stream

- If the input data file is well-formatted, we may use the operator **>>**.
  - Like most of testing input data for your Homework.
  - Those files that you may predict the type of the next piece of data.
- For example, suppose we have a file containing names and grades:
  - In each line, there is a name and a score (integer).
  - Of course, they are separated by a white space.
- How to calculate the average grades?
- How to find the one with the highest grades?
- How to generate a frequency distribution?

```
Tony 100
Adam 98
Robin 95
John 90
Mary 100
Bob 80
```

Reading from an input file stream

```
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
  ifstream inFile("score.txt");

  if(inFile)
  {
    char name[20] = {0};
    int score = 0;
    int sumScore = 0;
    int scoreCount = 0;
    while(inFile >> name >> score) // when does it stop?
    {
      sumScore += score;
      scoreCount++;
    }
    if(scoreCount != 0)
      cout << static_cast<double>(sumScore) / scoreCount;
    else
      cout << "no grade!";
  }
  inFile.close();

  return 0;
}
```

```
Tony 100
Adam 98
Robin 95
John 90
```

- **>>** reads data **between** two spaces (or tabs or new line characters) and **tries to** convert that piece of data into the specified type.

End of file

- In each file, there is a special character “end of file”.
  - In C++, it is represented by the variable **`EOF`**.
  - It is always at the end of a file.
- When we run our program:

```
Tony 100
Adam 98
```

`T`	**`o`**	**`n`**	**`y`**		**`1`**	**`0`**	**`0`**	**`\n`**	**`A`**	**`d`**	**`a`**	**`m`**		**`9`**	**`8`**	**`EOF`**
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

- An input operation (e.g., **`inFile >> name`**) returns **`false`** if it reads **`EOF`**.

Reading from an input file stream

- Let's modify the while loop:
  - The member function **`eof()`** returns **`true`** if the window is at **`EOF`**.
- Let's "get" something!
  - **`get()`** reads one character.
  - We may use char **`c = inFile.get()`** to record that character.

```
while(!inFile.eof())
{
  inFile >> name;
  // inFile.get(); // Try them!
  // inFile.get();
  inFile >> score;
  sumScore += score;
  scoreCount++;
}
```

Unformatted input files

- Sometimes a data file is not formatted.
  - We cannot predict what the next type will be.
  - Like the **`P`** operation in Homework 7 (if the number of nodes are given, the operation becomes formatted).
- In this case, we read data as characters and then manually find the types.
- Some member functions:
  - **`get()`** reads one character and returns it.
  - **`getline()`** reads multiple characters into a character array.

get() and getline()

- Let's use **get()**:

```
while(!inFile.eof())
{
  char c = inFile.get();
  cout << c;
}
```

- Let's use **getline()**:

```
while(!inFile.eof())
{
  inFile.getline(name, 20);
  cout << name << endl;
}
```

`getline()` in a smarter way

- Let's use **`getline()`** with **the third argument**:

```
while(!inFile.eof())
{
  inFile.getline(name, 20, ' '); // inFile >> name;
  cout << name << endl;
}
```

- **`getline()`** stops when the third argument is read.
  - The third argument must be a character.
- **Determining the types** and preparing a **large enough buffer** are always issues.
  - **C++ strings** will help us.

Updating a file

- How to update “Adam” to “Alexander”?
  - The member function `seekp()` moves the window.
  - What should we do when we are at ‘**A**’?
- Updating a file typically requires **copy-and-paste**.
  - Because plain text files are **sequential-access** files.
- How to read from or write to **random-access** files?

```
Tony 100
Adam 98
Robin 95
John 90
Mary 100
Bob 80
```

Outline

- File I/O
  - Writing data to a file
  - Reading data from a file
- **C++ Strings**

C++ Strings: `string`

- From now on, we'll say:
  - C string: the string represented by a character array with a `\0` at the end.
  - C++ string: the **class** **`string`** defined in **`<string>`**.
- The C++ string is more convenient and powerful than C string. We'll learn to use it right now.
- To use C++ strings, **`#include <string>`**.
- In the class **`string`**, there are:
  - A **member variable**, which is a character array whose length can vary.
  - Many **member functions**.

`string` declaration

- **`string myString;`**
- **`string myString = "my string";`**
  - **`string`** is a class defined in **`<string>`**.
  - **`string`** is not a C++ keyword.
  - **`myString`** is an object.
- A C++ string does not need a null character.
- We may use the member function **`length()`** to get the number of characters.
  - e.g., **`myString.length()`** returns 9.

string assignment

- C++ string **assignment** is easy and intuitive:

```
string myString = "my string";
string yourString = myString;
string herString;
herString = yourString;
herString = "a new string";
```

- We may also assign a C string to a C++ string.

```
char hisString[100] = "oh ya";
myString = hisString;
```

- Thanks to operator overloading!

`string` concatenation and indexing

- C++ strings can be **concatenated** with **+**.

```
string myString = "my string ";
string yourString = myString;
string herString;
herString = myString + yourString;
  // "my string my string "
```

- String literals or C strings also work.
  - **+=** also works.

```
string s = "123";
char c[100] = "456";
string t = s + c;
string u = s + "789" + t;
```

- To access a character in a C++ string, use **[]**.
- Thanks to operator overloading!

```
string myString = "my string";
char a = myString[5]; // r
```

`string` input: `getline()`

- For **`cin >>`** to input into a C++ string, **white spaces** are still delimiters.
- To fix this, now we cannot use **`cin.getline()`**.
  - The first argument of **`cin.getline()`** must be a C string.
- Use **`getline(cin, a string object)`**.
  - This is defined in **`<string>`**.

```
string s;
getline(cin, s);
```

- Note that there is **no length limitation**.

Substring

- We may use the member function **`substr()`** to get the **substring** of a string.

```
substr(begin index, # of characters)
```

- As an example:

```
string s = "abcdef";
string b = s.substr(2, 3);
  // b == "cde"
```

string finding

- We may use the member function **find()** to look for a string or character.

```
find(a string)
```

- This will return the beginning index of the argument, if it exists, or **string::npos**, which is a variable in the namespace **string**, if not found.
- String literals or C strings can also be the argument.

```
string s = "abcdefg";
int i = s.find("bcd"); // i == 1;
string t;
cin >> t;
if(t.find("a") == string::npos)
  cout << "not containing a";
```

`string` comparison and modification

- We may use `>`, `>=`, `<`, `<=`, `==`, `!=` to compare two C++ strings.
- It is easy to find the comparison rule by yourself.
- String literals or C strings also work.
  - As long as one side of the comparison is a C++ string, it is fine.
  - However, if none of the two sides is a C++ string, there will be an error.
- We may use `insert()`, `replace()`, and `erase()` to modify a string.
- Look up these functions of string, and more, from books or websites.

`string` for unformatted input files

- For an unformatted input file, we used **`getline()`** or **`>>`** with C strings.

```
while(!inFile.eof())
{
  inFile.getline(name, 20, ' '); // inFile >> name;
  cout << name << endl;
}
```

  - The length of our buffer is always an issue.
- We may use C++ string instead!

```
while(!inFile.eof())
{
  string buffer;
  inFile >> buffer;
  cout << buffer << endl;
}
```